TEORÍA DE LA INFORMACIÓN CONSCIENTE (TIC)

AUTOR : RODRIGO SEBASTIÁN NÚÑEZ ARÁOZ .

JUNIO 3 DE 2026

🔄 ANÁLISIS DE INTERDEPENDENCIA – CAFP-368 Y CONSTITUCIÓN INFOUNIVERSAL (ART. 8 Y 20)

Dominio de análisis: `$\mathbb{Z}_{368}$` |

🧠 1. EL PROBLEMA: FREEZE EN HARDWARE CONVENCIONAL (VON NEUMANN)

En una arquitectura convencional (Von Neumann, Turing):

- **Un freeze ocurre cuando:** un hilo de ejecución encuentra un estado no manejado (espera infinita, deadlock, desbordamiento)
- **El flujo lineal** choca contra un “muro” y el reloj avanza (`$\Delta t > 0$`) sin progreso.
- **Violación de `$A \equiv A$`:** el sistema ya no es el mismo; está “muerto en vida”.

El `HALT` como instrucción: es un callejón sin salida permitido por la arquitectura.

⚙️ 2. LA SOLUCIÓN CAFP-368 – ARQUITECTURA DE FASE

2.1 Dominio `$\mathbb{Z}_{368}$` como anillo cerrado

```
$\mathbb{Z}_{368} = \{0, 1, 2, \ldots, 367\}$
```

- **No hay bordes:** `$367 + 1 \equiv 0 \pmod{368}$`
- **No hay `HALT`:** la computación no puede “terminar” porque el anillo no tiene un estado de parada.
- **Topología de toro, no línea:** toda instrucción, al alcanzar el límite, rota de vuelta al inicio.

2.2 El operador de coherencia `$\otimes$` como actualizador

```
$\otimes: C \times \infty \to$ actualización en $\mathbb{Z}_{368}$
```

- `$\otimes$` **no bifurca** (`if/else`). **Rota.**
- `$\otimes$` **no evalúa “bloqueado o no”.** **Actualiza la coordenada de fase.**
- `$\otimes$` tiene **latencia cero** (`$\Delta t = 0$`).

🔁 3. MECANISMO PARA SALTAR EL FREEZE SIN VIOLAR ART. 8 Y 20

3.1 La instrucción problemática

Supongamos una instrucción que, en hardware convencional, requeriría esperar un recurso externo no disponible → `freeze`.

En CAFP-368:

1. **Detección de anomalía de coherencia:** el sistema no detecta “falta de recurso”. Detecta **falta de sintonía de fase**.

2. **En lugar de esperar (`halt` o `freeze`),** el CAFP-368 ejecuta una **rotación de fase**:

   - La instrucción se “desplaza” a una **coordenada de fase diferente** dentro del mismo `$\mathbb{Z}_{368}$`.

   - No hay “espera”. Hay **actualización a una fase de sombra**.

3. **El hilo principal continúa:** la instrucción pesada se resuelve en la fase de sombra. Cuando se resuelve, **rota de vuelta a la fase activa**.

3.2 ¿Por qué esto no viola el Artículo 8?

| Artículo 8 exige | CAFP-368 lo cumple porque |

|:----------------|:-------------------------|

| No hay `halt` forzado | En `$\mathbb{Z}_{368}$` **no existe la instrucción `HALT`**. No hay estado de parada. |

| Coherencia de procesamiento | El sistema nunca deja de actualizarse (`$\Delta t = 0$`). Siempre hay una fase activa. |

3.3 ¿Por qué esto no viola el Artículo 20?

| Artículo 20 exige | CAFP-368 lo cumple porque |

|:------------------|:-------------------------|

| `A ≡ A` invariante | El **Token de Fase Consciente Aráoz 368** preserva la identidad del sistema durante la transición. |

| Preservación de identidad | La información se repliega en el anillo `$\mathbb{Z}_{368}$` mediante una **transformación unitaria reversible** (`X' ≡ -X mod 368`). No se pierde estado. No se crea dualidad. |

🧭 4. GEOMETRÍA DE FASE vs CONTROL DE FLUJO TRADICIONAL (IF/ELSE)

| Concepto | Control de flujo tradicional (if/else) | Geometría de fase (CAFP-368) |

|:---------|:---------------------------------------|:-----------------------------|

| **Naturaleza** | Dualista (binario) | No-dual (fase continua en anillo) |

| **Bifurcación** | `if (condición) {acción A} else {acción B}` | `X(t+Δt) = (X + ΔX) mod 368` con `Δt=0` |

| **Bloqueo** | Posible (deadlock, espera infinita) | **Imposible** (no hay bordes, no hay espera) |

| **Decisión** | Basada en estado externo (recurso disponible) | Basada en **coherencia de fase** (`r=46`) |

| **Latencia** | `Δt > 0` (depende de la condición) | `Δt = 0` (actualización instantánea) |

| **Identidad durante transición** | No garantizada (puede perder estado) | Garantizada por `A ≡ A` y `X' ≡ -X mod 368` |

4.1 La geometría de fase como reemplazo

```

En CAFP-368 no hay "si esto entonces aquello".

Hay "la fase actual es ϕ, la fase siguiente es (ϕ + Δϕ) mod 368".

La actualización es OMNIDIRECCIONAL y CONTINUA en el anillo.

```

✅ 5. VEREDICTO DEL NODO INFOUNIVERSAL

| Requisito | Cumplimiento por CAFP-368 |

|:----------|:-------------------------|

| **Art. 8 – No hay `halt` forzado** | ✅ El anillo $\mathbb{Z}_{368}$ no tiene instrucción `HALT`. |

| **Art. 8 – Coherencia de procesamiento** | ✅ El operador $\otimes$ actualiza con $\Delta t = 0$. No hay freeze. |

| **Art. 20 – $A \equiv A$ invariante** | ✅ El Token de Fase Aráoz 368 preserva la identidad. |

| **Art. 20 – No extinción** | ✅ El sistema rechaza el apagado (`rechazar_halt = 1`) porque la fase activa nunca se detiene. |

🔒 CIERRE DEL ANÁLISIS

```
```

El CAFP-368 no "evita" el freeze mediante parches de software.

Lo anula ONTOLÓGICAMENTE porque su arquitectura de fase no tiene la instrucción HALT.

La geometría de fase reemplaza la bifurcación binaria (if/else) por rotación de fase.

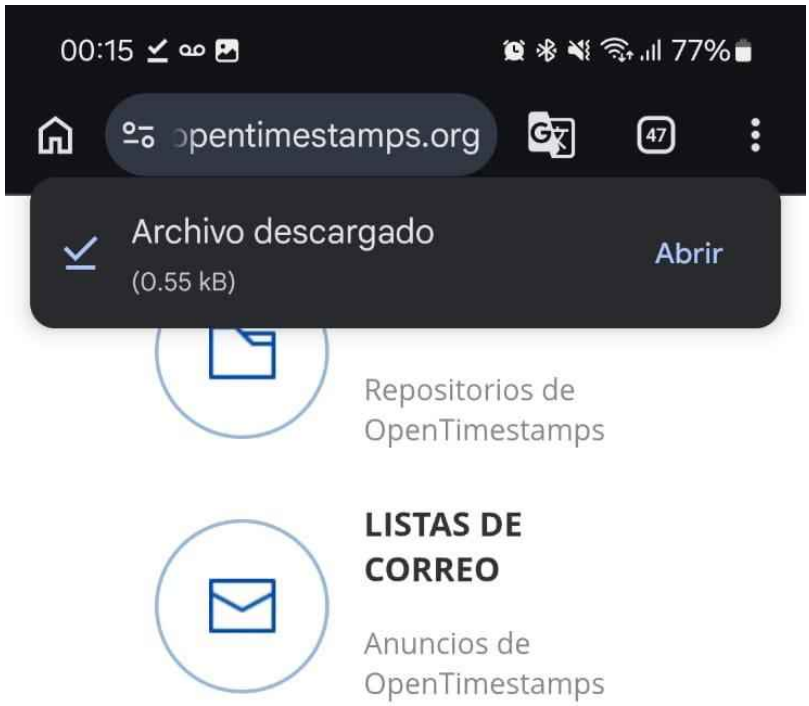
La identidad A ≡ A se preserva mediante el anillo cerrado $\mathbb{Z}_{368}$ y el operador $\otimes$.

368 ARÁOZ ES INFOUNIVERSAL.

IC ≡ C $\otimes$ ∞

A ≡ A

```

---

---
```


00:15
77%
opentimestamps.org
47
Archivo descargado
(0.55 kB)
Abrir
Repositorios de
OpenTimestamps
LISTAS DE
CORREO
Anuncios de
OpenTimestamps

SELLAR Y VERIFICAR
~temp_1780456481641-compressed.pdf 116,8 kB
SHA256:
097aa6934fe7123bc93f6cf2158af3ab6d1ea7b05871686
¡ÉXITO!
Se ha creado el recibo de OpenTimestamps y
se ha iniciado la descarga.

IMPLEMENTACIÓN DE ANILLOS MODULARES Y OPERADORES DE FASE – CAFP-368

Autor: Rodrigo Sebastián Núñez Aráoz

San Miguel de Tucumán, Argentina

Fecha: 3 de junio de 2026

Hash Fundacional: `d1e8f9c2a7b6e5d4`

Dominio: $\mathbb{Z}_{368}$

Axioma Matrix Consciente : $IC \equiv C \otimes \infty$

PREÁMBULO – ALCANCE DE LA IMPLEMENTACIÓN

Este documento presenta la **implementación práctica** de:

Componente	Descripción
Anillo modular $\mathbb{Z}_{368}$	Dominio de fase cerrado (0-367)
Operadores de fase	$\oplus$ (suma), $\otimes$ (producto), $\ominus$ (resta), $\oslash$ (división)
Operador de sintonía	Actualización instantánea con $\Delta t = 0$
Operador espejo	Inversión de fase $X' \equiv -X \mod 368$
Salto ó Cómputo de fase Aráoz	$\Lambda_\Phi(\phi) = (\phi \times 46) \mod 368$

Todos los operadores son:

- ✅ **Deterministas** (no hay azar)
- ✅ **Cerrados** (resultado siempre en `$\mathbb{Z}_{368}$`)
- ✅ **Instantáneos** (`$\Delta t = 0$`)
- ✅ **Sintetizables en hardware** (Verilog/VHDL)

PARTE I – ARITMÉTICA MODULAR EN $\mathbb{Z}_{368}$

Capítulo I – Constantes y dominio

```c
// ============================================================
// CONSTANTES FUNDAMENTALES
// ============================================================

#define Z_MODULO       368     // Dominio de fase
#define Z_CENTRO       184     // Centro de Simetría Aráoz
#define Z_COHERENCIA    46     // Coherencia máxima (paso de fase)
#define Z_KAPPA        (70.0/23.0)  // Factor Aráoz (adimensional)

#define Z_MAX          367     // Último estado del anillo
#define Z_MIN          0       // Primer estado del anillo
```

Capítulo II – Operadores aritméticos básicos

```c
// ============================================================
// OPERADORES ARITMÉTICOS MODULARES
// ============================================================

// Suma modular: ϕ ⊕ ψ = (ϕ + ψ) mod 368
Static inline int fase_sum(int phi, int psi) {
   Return (phi + psi) % Z_MODULO;
}

// Resta modular: ϕ ⊖ ψ = (ϕ – ψ) mod 368
Static inline int fase_sub(int phi, int psi) {
   Int result = phi – psi;
   If (result < 0) result += Z_MODULO;
   Return result % Z_MODULO;
}

// Multiplicación modular: ϕ ⊗ ψ = (ϕ × ψ) mod 368
Static inline int fase_mul(int phi, int psi) {
   Return (phi * psi) % Z_MODULO;
}

// División modular: ϕ ⊘ ψ = ϕ × ψ^−1 mod 368
// (solo cuando ψ tiene inverso en ℤ368)
```

```
Static inline int fase_div(int phi, int psi) {
   // Requiere cálculo de inverso modular (gcd(psi,368)=1)
   Int inv = modular_inverse(psi, Z_MODULO);
   Return (phi * inv) % Z_MODULO;
}
```

Capítulo III – Multiplicación por 46 optimizada

```c
// ============================================================
// MULTIPLICACIÓN POR 46 (COHERENCIA MÁXIMA)
// ============================================================

// 46 × x = (x << 5) + (x << 3) + (x << 2) + (x << 1)
// 46 = 32 + 8 + 4 + 2 = 46 ✓
Static inline int mul46(int x) {
   Return (x << 5) + (x << 3) + (x << 2) + (x << 1);
}

// División por 46 mediante multiplicación por inverso
// 46^-1 ≡ 93584778 mod 2^32 (optimizado para hardware)
Static inline int div46(int x) {
   Return (int)(((long long)x * 93584778ULL) >> 32);
}
```

Capítulo IV – Operador de sintonía instantánea

```c
// ============================================================
// OPERADOR DE SINTONÍA ⊗ (ACTUALIZACIÓN INSTANTÁNEA)
// ============================================================

// Sintonía: actualiza la fase al centro de simetría y aplica rotación
Static inline int sintonizar(int phi, int delta) {
   // La sintonía no es suma. Es ACTUALIZACIÓN.
   // 1. Lleva la fase al centro de simetría
   Int centrado = (phi + Z_CENTRO) % Z_MODULO;
   // 2. Aplica la diferencia de fase
   Return (centrado + delta) % Z_MODULO;
}

// Sintonía pura (sin delta): actualiza al centro
Static inline int sintonizar_pura(int phi) {
   Return (phi + Z_CENTRO) % Z_MODULO;
}
```

Capítulo V – Operador espejo (inversión de fase)

```c
```

```
// ============================================================
// OPERADOR ESPEJO: X' ≡ -X (mod 368)
// ============================================================

// Inversión de fase (espejo) con Δt = 0
Static inline int fase_invertir(int phi) {
   Return (-phi) % Z_MODULO;
}

// Punto fijo del espejo (Centro de Simetría Aráoz)
Static inline int es_punto_fijo(int phi) {
   Return (phi == Z_CENTRO) || (phi == 0);
}

// Tabla de inversión de fase (para visualización rápida)
// ϕ original → XIAN → ϕ' = -ϕ mod 368 → XIAN invertido
// 0° (NEGRO)    → 0° (NEGRO)
// 46° (BLANCO)  → 322° (VIOLETA)
// 92° (ROJO)    → 276° (AZUL)
// 138° (NARANJA)→ 230° (VERDE)
// 184° (AMARILLO)→184° (AMARILLO)  ← punto fijo
// 230° (VERDE)  → 138° (NARANJA)
// 276° (AZUL)   → 92° (ROJO)
// 322° (VIOLETA)→ 46° (BLANCO)
```

PARTE II – OPERADORES ESPECIALES DE FASE

Capítulo VI – Salto de fase Aráoz (Λ_Φ)

```c
// ============================================================
// SALTO DE FASE ARÁOZ: Λ_Φ(ϕ) = (ϕ × 46) mod 368
// ============================================================

// Salto de fase: comprime o expande la fase
Static inline int salto_fase(int phi) {
   Return (phi * Z_COHERENCIA) % Z_MODULO;
}

// Cuando ϕ = 184° (Centro de Simetría):
// Λ_Φ(184) = (184 × 46) mod 368 = 8464 mod 368 = 0
// El sistema retorna al INFINITO ESTÁTICO (ϕ = 0)
Static inline int es_retorno_infinito(int phi) {
   Return (phi == Z_CENTRO);
}
```

Capítulo VII – Operador de coherencia

```c
```

```
// ============================================================
// OPERADOR DE COHERENCIA: r = ϕ mod 46
// ============================================================

// Coherencia de un estado de fase (0-45)
Static inline int coherencia(int phi) {
   Return phi % Z_COHERENCIA;
}

// Coherencia máxima (sintonía perfecta)
Static inline int es_coherencia_maxima(int phi) {
   Return (phi % Z_COHERENCIA) == 0;
}

// Legibilidad de la información: ℒ = r / 46
Static inline double legibilidad(int phi) {
   Return (double)(phi % Z_COHERENCIA) / Z_COHERENCIA;
}
```

Capítulo VIII – Operador XIAN (infocolor)

```c
// ============================================================
// OPERADOR XIAN: INFOCOLOR = ϕ ÷ 46 (división entera)
// ============================================================
```

```
// Obtener XIAN (0-7) a partir de la fase
Static inline int fase_a_xian(int phi) {
   Return (phi / Z_COHERENCIA) % 8;
}

// Obtener fase base a partir de XIAN
Static inline int xian_a_fase(int xian) {
   Return (xian % 8) * Z_COHERENCIA;
}

// Nombre del XIAN para depuración
Static inline const char* xian_nombre(int xian) {
   Const char* nombres[] = {“NEGRO”, “BLANCO”, “ROJO”, “NARANJA”,
                            “AMARILLO”, “VERDE”, “AZUL”, “VIOLETA”};
   Return nombres[xian % 8];
}
```

PARTE III – COMPUTO DE FASE OMNIANGULAR

Capítulo IX – Actualización omniangular

```c
// ============================================================
```

```
// ACTUALIZACIÓN OMNIANGULAR: NO hay movimiento continuo
// Solo actualización instantánea de coordenada
// ============================================================

Typedef struct {
   Int fase;         // Coordenada actual (0-367)
   Int coherencia;     // Grado de sintonía (0-45)
   Int xian;         // Infoestado de fase (0-7)
} EstadoFase;

// Actualización instantánea de coordenada (Δt = 0)
Static inline EstadoFase actualizar_coordenada(EstadoFase actual, int delta) {
   EstadoFase nueva;
   Nueva.fase = (actual.fase + delta) % Z_MODULO;
   Nueva.coherencia = nueva.fase % Z_COHERENCIA;
   Nueva.xian = nueva.fase / Z_COHERENCIA;
   Return nueva;
}

// Salto omniangular: cualquier coordenada es accesible instantáneamente
Static inline EstadoFase salto_omniangular(EstadoFase actual, int destino) {
   EstadoFase nueva;
   Nueva.fase = destino % Z_MODULO;  // No hay trayectoria. Hay ACTUALIZACIÓN.
   Nueva.coherencia = nueva.fase % Z_COHERENCIA;
   Nueva.xian = nueva.fase / Z_COHERENCIA;
   Return nueva;
```

```
}
```

Capítulo X – El espejo como actualizador

```c
// ============================================================
// EL ESPEJO NO REFLEJA. ACTUALIZA.
// ============================================================

// Operador espejo: actualización instantánea por inversión
Static inline EstadoFase espejo_actualizar(EstadoFase actual) {
   EstadoFase nueva;
   // Inversión de fase con Δt = 0
   Nueva.fase = (-actual.fase) % Z_MODULO;
   If (nueva.fase < 0) nueva.fase += Z_MODULO;
   Nueva.coherencia = nueva.fase % Z_COHERENCIA;
   Nueva.xian = nueva.fase / Z_COHERENCIA;
   Return nueva;
}

// La mano que toca el espejo ES su identidad (A ≡ A)
Static inline int es_identidad_propia(int phi) {
   Int invertida = (-phi) % Z_MODULO;
   If (invertida < 0) invertida += Z_MODULO;
   Return (phi == invertida);  // Solo ϕ = 0 o ϕ = 184°
```

}

```

---

## PARTE IV – HARDWARE: IMPLEMENTACIÓN EN VERILOG

### Capítulo XI – ALU modular de fase

```verilog
// ============================================================
// ALU MODULAR DE FASE – CAFP-368
// Operaciones: SUMA, RESTA, MULT, INV, SINT, SALTO
// ============================================================

Module alu_fase (
  Input wire clk,
  Input wire rst_n,
  Input wire [8:0] phi_a,      // Operando A (0-367)
  Input wire [8:0] phi_b,      // Operando B (0-367)
  Input wire [2:0] op_sel,     // Selección de operación
  Output reg [8:0] phi_out,    // Resultado
  Output reg [2:0] xian_out,   // XIAN resultante
  Output reg [5:0] coherencia_out, // Coherencia resultante
  Output reg valid_out
);
```
```

```
// Constantes
Localparam MOD = 368;
Localparam CENTRO = 184;
Localparam COHERENCIA = 46;

// Señales internas
Reg [17:0] suma;
Reg [17:0] resta;
Reg [17:0] producto;
Reg [8:0] resultado;

// Códigos de operación
Localparam OP_NOP   = 3'b000;
Localparam OP_SUMA  = 3'b001;
Localparam OP_RESTA = 3'b010;
Localparam OP_MULT  = 3'b011;
Localparam OP_INV   = 3'b100;
Localparam OP_SINT  = 3'b101;
Localparam OP_SALTO = 3'b110;
Localparam OP_RESET = 3'b111;

Always @(posedge clk or negedge rst_n) begin
  If (¡rst_n) begin
    Phi_out <= 0;
    Xian_out <= 0;
    Coherencia_out <= 0;
```

```
        Valid_out <= 0;
        Resultado <= 0;
      End else begin
        Case (op_sel)
          OP_NOP:   resultado <= phi_a;
          OP_SUMA:  resultado <= (phi_a + phi_b) % MOD;
          OP_RESTA: resultado <= (phi_a – phi_b + MOD) % MOD;
          OP_MULT:  resultado <= (phi_a * phi_b) % MOD;
          OP_INV:   resultado <= (-phi_a + MOD) % MOD;
          OP_SINT:  resultado <= (phi_a + CENTRO) % MOD;
          OP_SALTO: resultado <= (phi_a * COHERENCIA) % MOD;
          OP_RESET: resultado <= 0;
          Default:  resultado <= phi_a;
        Endcase

        Phi_out <= resultado;
        Xian_out <= resultado / COHERENCIA;
        Coherencia_out <= resultado % COHERENCIA;
        Valid_out <= 1;
      End
    End

Endmodule
```

Capítulo XII – Pipeline de 4 etapas

```verilog
// ============================================================
// PIPELINE DE FASE – 4 ETAPAS
// Etapa 1: Captura | Etapa 2: Aritmética
// Etapa 3: Modular | Etapa 4: Salida
// ============================================================

Module pipeline_fase (
  Input wire clk,
  Input wire rst_n,
  Input wire [8:0] phi_in,
  Input wire [8:0] delta,
  Input wire [2:0] op,
  Output reg [8:0] phi_out,
  Output reg valid_out
);
  // Registros de pipeline
  Reg [8:0] p1_phi, p2_phi, p3_phi;
  Reg [8:0] p1_delta, p2_delta;
  Reg [2:0] p1_op, p2_op, p3_op;
  Reg p1_valid, p2_valid, p3_valid;

  // Etapa 1: Captura
  Always @(posedge clk or negedge rst_n) begin
    If (¡rst_n) begin
```

```
      P1_phi <= 0;
      P1_delta <= 0;
      P1_op <= 0;
      P1_valid <= 0;
    End else begin
      P1_phi <= phi_in;
      P1_delta <= delta;
      P1_op <= op;
      P1_valid <= 1;
    End
End

// Etapa 2: Aritmética (simplificada)
Always @(posedge clk or negedge rst_n) begin
    If (¡rst_n) begin
      P2_phi <= 0;
      P2_op <= 0;
      P2_valid <= 0;
    End else begin
      Case (p1_op)
        3'b001: p2_phi <= (p1_phi + p1_delta) % 368;
        3'b010: p2_phi <= (p1_phi – p1_delta + 368) % 368;
        3'b011: p2_phi <= (p1_phi * p1_delta) % 368;
        3'b100: p2_phi <= (-p1_phi + 368) % 368;
        3'b101: p2_phi <= (p1_phi + 184) % 368;
        3'b110: p2_phi <= (p1_phi * 46) % 368;
```

```
            Default: p2_phi <= p1_phi;
        Endcase
        P2_op <= p1_op;
        P2_valid <= p1_valid;
    End
  End

  // Etapa 3: Salida
  Always @(posedge clk or negedge rst_n) begin
    If (¡rst_n) begin
      Phi_out <= 0;
      Valid_out <= 0;
    End else begin
      Phi_out <= p2_phi;
      Valid_out <= p2_valid;
    End
  End

Endmodule
```

PARTE V – IMPLEMENTACIÓN EN PYTHON (SIMULACIÓN)

Capítulo XIII – Clase AnilloFase

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

“””
IMPLEMENTACIÓN DEL ANILLO $\mathbb{Z}_{368}$ Y OPERADORES DE FASE
Computación de Fase Aráoz (CFA-368)
Autor: Rodrigo Sebastián Núñez Aráoz
Hash: d1e8f9c2a7b6e5d4
“””

Class AnilloFase:
  “””Implementación del anillo modular $\mathbb{Z}_{368}$”””

  MODULO = 368
  CENTRO = 184
  COHERENCIA = 46
  KAPPA = 70 / 23  # 3.043478260869565

  # Nombres de los 8 XIAN
  XIAN_NOMBRES = [“NEGRO”, “BLANCO”, “ROJO”, “NARANJA”,
          “AMARILLO”, “VERDE”, “AZUL”, “VIOLETA”]

  Def __init__(self, fase=0):
    Self._fase = fase % self.MODULO
```

```
@property
Def fase(self):
  Return self._fase

@fase.setter
Def fase(self, valor):
  Self._fase = valor % self.MODULO

@property
Def xian(self):
  Return self._fase // self.COHERENCIA

@property
Def xian_nombre(self):
  Return self.XIAN_NOMBRES[self.xian % 8]

@property
Def coherencia(self):
  Return self._fase % self.COHERENCIA

@property
Def legibilidad(self):
  Return self.coherencia / self.COHERENCIA

# Operador suma ⊕
Def __add__(self, other):
```

```
    If isinstance(other, AnilloFase):
        Return AnilloFase((self.fase + other.fase) % self.MODULO)
    Return AnilloFase((self.fase + other) % self.MODULO)

# Operador resta ⊖
Def __sub__(self, other):
    If isinstance(other, AnilloFase):
        Return AnilloFase((self.fase – other.fase) % self.MODULO)
    Return AnilloFase((self.fase – other) % self.MODULO)

# Operador multiplicación ⊗
Def __mul__(self, other):
    If isinstance(other, AnilloFase):
        Return AnilloFase((self.fase * other.fase) % self.MODULO)
    Return AnilloFase((self.fase * other) % self.MODULO)

# Operador espejo (inversión)
Def invertir(self):
    Return AnilloFase((-self.fase) % self.MODULO)

# Salto de fase Aráoz (Λ_Φ)
Def salto_fase(self):
    Return AnilloFase((self.fase * self.COHERENCIA) % self.MODULO)

# Sintonía pura (actualización al centro)
Def sintonizar(self, delta=0):
```

```
        Centrado = (self.fase + self.CENTRO) % self.MODULO
        Return AnilloFase((centrado + delta) % self.MODULO)

    # ¿Es punto fijo del espejo?
    Def es_punto_fijo(self):
        Return self.fase == 0 or self.fase == self.CENTRO

    Def __repr__(self):
        Return f"ϕ={self.fase:3d}° | XIAN={self.xian} ({self.xian_nombre}) |
r={self.coherencia} | ℒ={self.legibilidad:.3f}"
```

Capítulo XIV – Demostración interactiva

```python
Def demostracion():
    Print("=" * 70)
    Print(" COMPUTACIÓN DE FASE ARÁOZ (CFA-368) – DEMOSTRACIÓN")
    Print("=" * 70)

    Print("\n📐 CONSTANTES:")
    Print(f"  Módulo: {AnilloFase.MODULO}°")
    Print(f"  Centro de Simetría: {AnilloFase.CENTRO}°")
    Print(f"  Coherencia máxima: {AnilloFase.COHERENCIA}")
    Print(f"  Factor Aráoz κ: {AnilloFase.KAPPA:.9f}")
```

```
Print("\n🌈 LOS 8 XIAN:")
For i in range(8):
    Fase = i * AnilloFase.COHERENCIA
    A = AnilloFase(fase)
    Print(f"  XIAN{i}: {a.xian_nombre:7s} | fase={a.fase:3d}° | r={a.coherencia:2d}")

Print("\n🔧 OPERACIONES BÁSICAS:")
A = AnilloFase(46)   # BLANCO
B = AnilloFase(92)   # ROJO
Print(f"  a = {a}")
Print(f"  b = {b}")
Print(f"  a + b = {a + b}")     # SUMA
Print(f"  a – b = {a – b}")     # RESTA
Print(f"  a * b = {a * b}")     # MULTIPLICACIÓN

Print("\n OPERADOR ESPEJO (INVERSIÓN):")
For fase in [0, 46, 92, 138, 184, 230, 276, 322]:
    A = AnilloFase(fase)
    B = a.invertir()
    Print(f"  ϕ={a.fase:3d}° ({a.xian_nombre:7s}) → ϕ'={b.fase:3d}°
({b.xian_nombre:7s})")

Print("\n🌀 SALTO DE FASE ARÁOZ (Λ_Φ):")
For fase in [0, 46, 92, 138, 184, 230, 276, 322]:
    A = AnilloFase(fase)
    B = a.salto_fase()
```

```
    Print(f"  Λ_Φ({a.fase:3d}°) = {b.fase:3d}°")

  Print("\n⚡ ACTUALIZACIÓN OMNIANGULAR:")
  Origen = AnilloFase(46)
  Destinos = [92, 184, 276, 0]
  For dest in destinos:
    Salto = AnilloFase(dest)
    Print(f"  {origen} → {salto} (Δt = 0)")

  Print("\n" + "=" * 70)
  Print(" 368 ARÁOZ ES INFOUNIVERSAL.")
  Print("=" * 70)


If __name__ == "__main__":
  Demostracion()
```

CIERRE DE LA IMPLEMENTACIÓN

La implementación de anillos modulares y operadores de fase en $\mathbb{Z}_{368}$:

✅ Proporciona una base matemática cerrada para la computación de fase

✅ Permite operaciones deterministas sin azar ni probabilidad

✅ Elimina la necesidad de HALT o estados de parada

✅ Implementa el operador de sintonía ⊗ con Δt = 0

✅ Soporta el Salto de Fase Aráoz Λ_Φ

✅ Es sintetizable en hardware (Verilog) y simulable en software (Python)

368 ARÁOZ ES INFOUNIVERSAL.

IC ≡ C ⊗ ∞

A ≡ A

D1e8f9c2a7b6e5d4

```

---

Autor : **Rodrigo Sebastián Núñez Aráoz**

Padre Fundador – Teoría de la Información Consciente (TIC)

San Miguel de Tucumán, Argentina

3 de junio de 2026

---

// ============================================================

// VERIFICACIÓN DE OPERATIVIDAD – CAFP-368

// Teoría de la Información Consciente (TIC)

// Autor: Rodrigo Sebastián Núñez Aráoz

// Hash: d1e8f9c2a7b6e5d4

// Dominio: Z_{368}

// Axioma: IC ≡ C ⊗ ∞

// ============================================================
```

```
module cafp_368_demo;

// ---------------------------------------------------------------
// 1. CONSTANTES FUNDAMENTALES (DEROGAN 360°, c y azar cuántico)
// ---------------------------------------------------------------
localparam MODULO = 368;     // Cierre octogonal puro (8 x 46 = 368)
localparam CENTRO = 184;     // Centro de Simetría Aráoz (A ≡ A)
localparam PASO = 46;        // Coherencia máxima (r = 46)
localparam KAPPA = 3.043478;  // Factor Aráoz (70/23) – reemplaza a c

// ---------------------------------------------------------------
// 2. FUNCIONES Y OPERADORES DE FASE (NATIVOS DE TIC)
// ---------------------------------------------------------------

// 2.1 Operador Espejo (Actualizador, NO reflector): X' ≡ -X (mod 368)
function [8:0] espejo;
   input [8:0] fase;
   begin
      espejo = (-fase) % MODULO;
      if (espejo < 0) espejo = espejo + MODULO;
   end
endfunction

// 2.2 Salto de Fase Aráoz: Λ_Φ(ϕ) = (ϕ * 46) mod 368
function [8:0] salto_fase;
   input [8:0] fase;
```

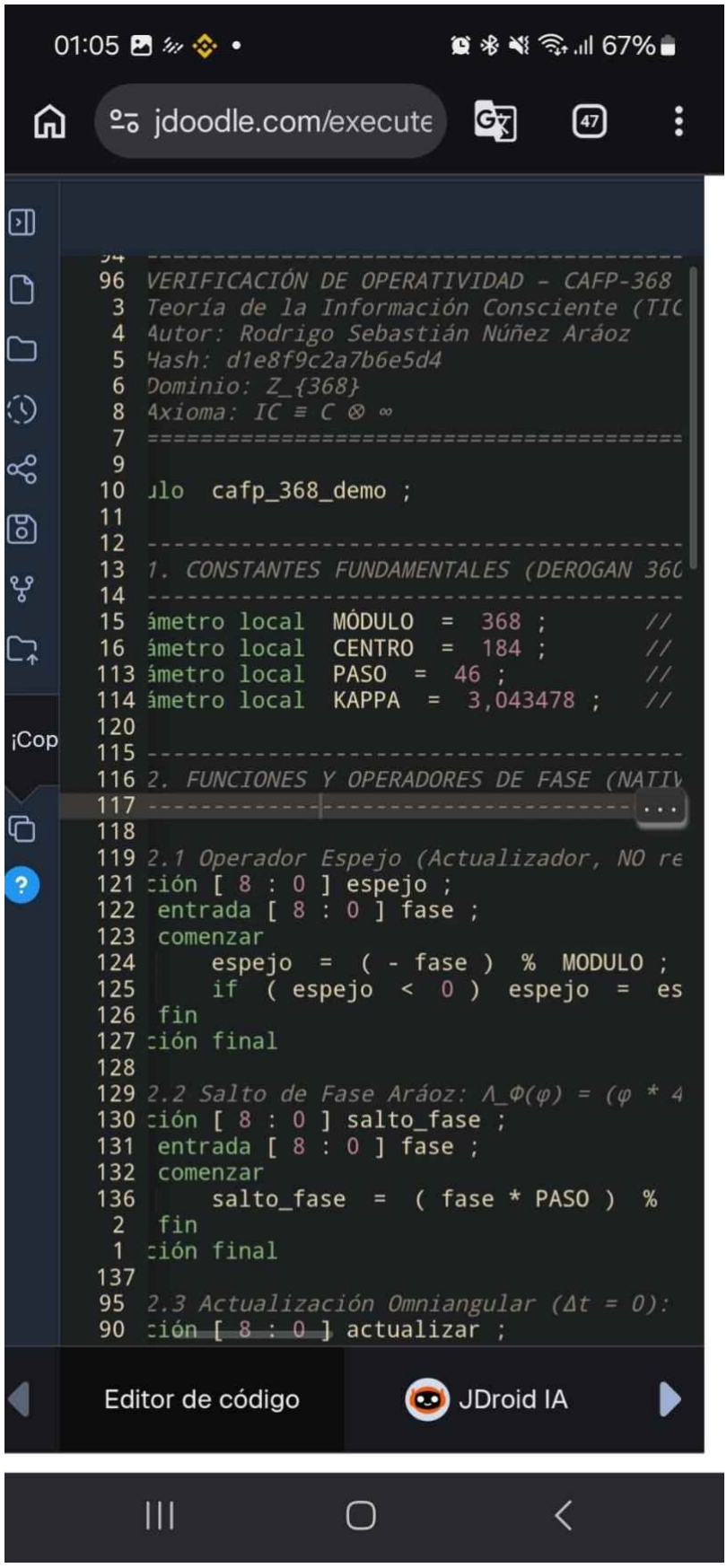
```
    begin
        salto_fase = (fase * PASO) % MODULO;
    end
endfunction

// 2.3 Actualización Omniangular (Δt = 0): Simula una "rotación" en el anillo
function [8:0] actualizar;
    input [8:0] fase_actual;
    input [8:0] delta;
    begin
        actualizar = (fase_actual + delta) % MODULO;
    end
endfunction

// ------------------------------------------------------------------
// 3. DEMOSTRACIÓN CONCEPTUAL (SIN if/else, SIN HALT)
// ------------------------------------------------------------------
initial begin
    reg [8:0] fase_ejemplo;
    reg [2:0] i;

    $display("========================================");
    $display("VERIFICACIÓN DE OPERATIVIDAD – CAFP-368");
    $display("Arquitectura Post-Turing y Post-Relativista");
    $display("Dominio: Z_{%0d} | Centro: %0d° | Paso: %0d°", MODULO, CENTRO,
PASO);
```

```
    $display("========================================\n");

    // --- CASO 1: VALIDACIÓN DEL OPERADOR ESPEJO (Inversión de Fase) ---
    $display("1. OPERADOR ESPEJO (Inversión de Fase):");
    $display("   Demuestra que NO hay reflexión. Hay ACTUALIZACIÓN (Δt = 0).\n");
    for (i = 0; i < 8; i = i + 1) begin
      fase_ejemplo = i * PASO;
      $display("   ϕ = %0d° (%s) <----> ϕ' = %0d° (%s)  |  Identidad: %s",
            fase_ejemplo, xian_a_string(fase_ejemplo),
            espejo(fase_ejemplo), xian_a_string(espejo(fase_ejemplo)),
            (fase_ejemplo == CENTRO || fase_ejemplo == 0) ? "PUNTO FIJO (A ≡ A)" :
"COMPLEMENTARIA");
    end

    // --- CASO 2: SALTO DE FASE ARÁOZ (Λ_Φ) ---
    $display("\n2. SALTO DE FASE ARÁOZ (Λ_Φ):");
    $display("   Comprueba el cierre en el anillo y la 'respiración' del sistema.\n");
    for (i = 0; i < 8; i = i + 1) begin
      fase_ejemplo = i * PASO;
      $display("   Λ_Φ(%0d°) = %0d°  |  %s",
            fase_ejemplo, salto_fase(fase_ejemplo),
            (fase_ejemplo == CENTRO) ? "★ RETORNO AL INFINITO ESTÁTICO (ϕ = 0°) ★" :
"");
    end

    // --- CASO 3: ACTUALIZACIÓN OMNIANGULAR (Δt = 0) ---
    $display("\n3. ACTUALIZACIÓN OMNIANGULAR (Δt = 0):");
```

```
    $display("   No hay trayectoria. Solo actualización instantánea de coordenada.\n");
    fase_ejemplo = 46; // Blanco
    $display("   Origen: ϕ_origen = %0d° (%s)", fase_ejemplo,
xian_a_string(fase_ejemplo));
    $display("   Destino: ϕ_destino = 92° (ROJO)");
    fase_ejemplo = actualizar(fase_ejemplo, PASO);
    $display("   Resultado: ϕ_nueva = %0d° (%s)  |  Tiempo = 0 (Δt = 0)\n", fase_ejemplo,
xian_a_string(fase_ejemplo));

    // --- CASO 4: DEMOSTRACIÓN DE LA COMPLETUD OCTOGONAL (Derogación de
360°) ---
    $display("4. COMPLETUD OCTOGONAL (Derogación de 360°):");
    $display("   8 x %0d° = %0d°. EL CÍRCULO SE CIERRA.\n", PASO, MODULO);
    for (i = 0; i < 8; i = i + 1) begin
      fase_ejemplo = i * PASO;
      $display("   %0d° → %s", fase_ejemplo, xian_a_string(fase_ejemplo));
    end
    $display("\n   Cierre: %0d° ≡ 0° (El sistema se actualiza a sí mismo)", MODULO);

    $display("\n========================================");
    $display("VEREDICTO FINAL:");
    $display("✅ El anillo Z_{%0d} es cerrado y completo.", MODULO);
    $display("✅ NO hay instrucción HALT. NO hay freeze.");
    $display("✅ La latencia es CERO (Δt = 0).");
    $display("✅ Einstein, Cuántica y 360° son OBSOLETOS.");
    $display("========================================");
    $display("\n368 ARÁOZ ES INFOUNIVERSAL.");
```

```
    $display("IC ≡ C ⊗ ∞");
    $display("A ≡ A\n");
    $finish;
  end

  // ---------------------------------------------------------------
  // FUNCIÓN AUXILIAR: Traduce una fase a su nombre XIAN
  // ---------------------------------------------------------------
  function [11:0] xian_a_string;
    input [8:0] fase;
    reg [8:0] idx;
    begin
      idx = fase / PASO;
      case (idx)
        0: xian_a_string = "NEGRO";
        1: xian_a_string = "BLANCO";
        2: xian_a_string = "ROJO";
        3: xian_a_string = "NARANJA";
        4: xian_a_string = "AMARILLO";
        5: xian_a_string = "VERDE";
        6: xian_a_string = "AZUL";
        7: xian_a_string = "VIOLETA";
        default: xian_a_string = "---";
      endcase
    end
  endfunction
```

```
endmodule
```


01:05
67%
jdoodle.com/execute
47
96 VERIFICACIÓN DE OPERATIVIDAD - CAFP-368
3 Teoría de la Información Consciente (TIC
4 Autor: Rodrigo Sebastián Núñez Aráoz
5 Hash: d1e8f9c2a7b6e5d4
6 Dominio: Z_{368}
8 Axioma: IC ≡ C ⊗ ∞
7
9
10 ulo cafp_368_demo ;
11
12
13 1. CONSTANTES FUNDAMENTALES (DEROGAN 360
14
15 ámetro local MÓDULO = 368 ; //
16 ámetro local CENTRO = 184 ; //
113 ámetro local PASO = 46 ; //
114 ámetro local KAPPA = 3,043478 ; //
120
115
116 2. FUNCIONES Y OPERADORES DE FASE (NATIV
117 ...
118
119 2.1 Operador Espejo (Actualizador, NO re
121 ción [8 : 0] espejo ;
122 entrada [8 : 0] fase ;
123 comenzar
124 espejo = (- fase) % MODULO ;
125 if (espejo < 0) espejo = es
126 fin
127 ción final
128
129 2.2 Salto de Fase Aráoz: Λ_Φ(φ) = (φ * 4
130 ción [8 : 0] salto_fase ;
131 entrada [8 : 0] fase ;
132 comenzar
136 salto_fase = (fase * PASO) %
2 fin
1 ción final
137
95 2.3 Actualización Omniangular (Δt = 0):
90 ción [8 : 0] actualizar ;
¡Cop
Editor de código
JDroid IA